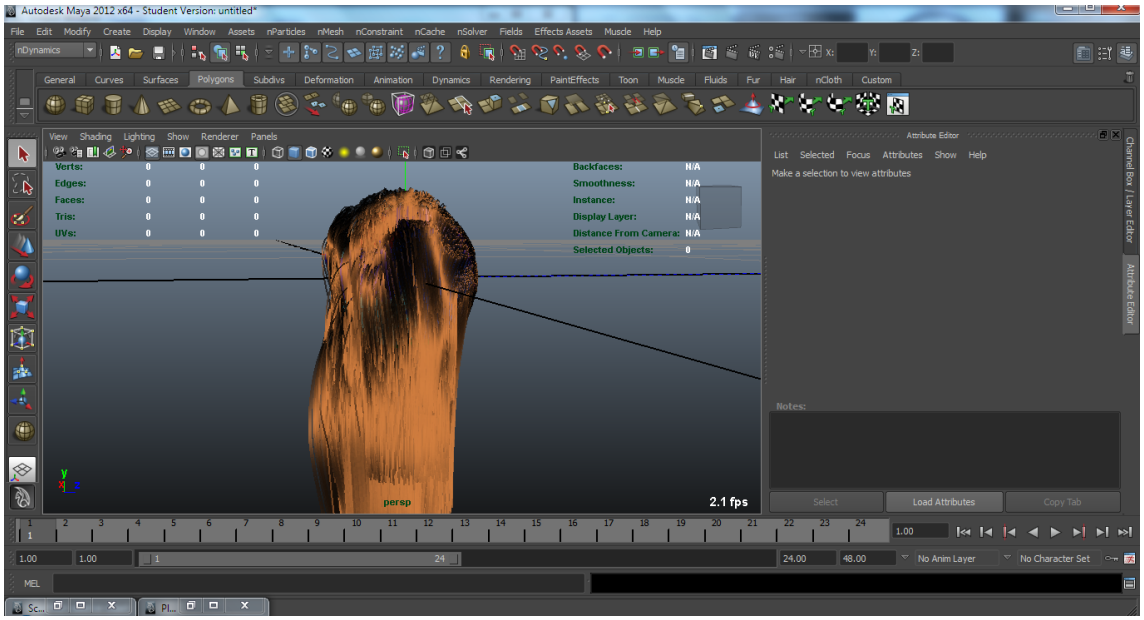
Hair generation and shading Maya plug-in

SFX tricks of the trade

Dan Englesson
Toms Vulfs

January 20, 2012



Abstract

Hair in computer graphics can be simulated and rendered with very high quality but also at high rendering times. There are several methods that produce very good visual results in real-time applications for both hair simulations and shading. This report presents some of these methods such as Marschner, Kajiya-Kay shading models, hair generation and other models in detail which are combined into a Maya plug-in that renders the generated hair in real-time in the viewport.

1 Introduction

Simulating hair on characters in the visual effects industry is becoming more and more demanding due to closeup scenes whole characters covered in hair and visual effects hair interaction with humans in real-life shots. Animated movies like Tangled, How to train a dragon and many other blockbuster movie titles where small furry creatures are shown all use some kind of hair simulation to give a more visual appealing appearance to them. Many tool are therefore available at the visual effects companies. Artist often have to be experts on these sort of tools since real-time response of hair simulations are not available. They have to rely on manually placed guide hairs to determine the quality and look of the resulting hairs or by rendering single frames over and over to adjust the style. This report covers some the hair generation and shading methods that can be used in such an real-time tool, and these methods are then implemented on Autodesk Maya in the viewport.

In section 2 the underlying method for the hair implementation of this paper is presented. Sections 2 and 3 describe the implementation details followed by section 4 where an explanation is given of the approach of moving the implementation into a Maya plug-in. Lastly results and a discussion are presented in section 5 respectively 6.

2 Method

This section explains the governing methods used in the implementation

2.1 Generating hair

The procedure of creating hairs on meshes is done by first generating random points on the mesh. To ensure that the guide points are placed in optimal distances between each other a Centroidal Voronoi Tessellation(CVT) [1] scheme is used. It is possible to ensure that the guide hairs are uniformly placed by distributing the guide hair points using CVT. In the cases where an artist places the guide hairs on a mesh manually, the interpolated hairs can get unwanted behavior since the distance between them is uneven. Given a large character like the wolves in the Twilight saga movies guide hairs can be of a magnitude of 4000 and to place each and everyone so that they influence the interpolated hairs the right way can be a difficult task, even for a skilled artist.

Hair strands are built from Non-Uniform Rational B-Splines (NURBS) which provide flexibility and precision for evaluating analytic curves. To obtain many hairs on a surface from the given guide hairs a single-strand interpolation scheme is used which computes a hair curve from its closest guide hair neighbors. Each of these procedures mentioned are explained throughly in the following subsections.

2.2 Nurbs curves

NURBS curves are derived from parametric curves which are built from a set of coefficients and basis functions. Such a curve can be built from only a few control points forming simple line segments. To form a smooth hair curve from the control points of a strand a discrete number of points are used. The number of points defines the smoothness level of the resulting curve and an number of 41 is used in this paper, yielding good results. A NURBS curve is defined by:

$$B(t) = \frac{\sum_n^{i=0} N_{i,p}(t) w_i P_i}{\sum_n^{i=0} N_{i,p} w_i} \qquad (1)$$

In equation 1 P_i are the control points, $N_{i,p}$ are the B-spline basis functions and w_i are weights for additional control of the curve shape. For this paper weights where set to 1 since no additional controls are required. One of the advantages for B-spline basis functions is that they can be defined by a recursive function with respect to a

knot vector $\mathbf{t}_i$ which determines the type of curve. The basis function is defined by:

$$N_{i,0} = \begin{cases} 1 & if \mathbf{t}_i \leq t < \mathbf{t}_{i+1} \\ 0 & otherwise \end{cases} \quad (2)$$

$$N_{i,p}(t) = \frac{t-\mathbf{t}_i}{\mathbf{t}_{i+p}-\mathbf{t}_i} N_{i,p-1}(t) + \frac{\mathbf{t}_{i+p+1}-t}{\mathbf{t}_{i+p+1}-\mathbf{t}_{i+1}} N_{i+1,p-1}(t) \quad (3)$$

A B-spline curve can also be defined by curve subdivision which implies that a successive application of subdivision of the control points where except the boundaries there are only two rules. Every control point p_i is re-wighted by equations 4 and 5 thus placed between two old ones where the index $\frac{1}{2}$ refers to newly created control point.

$$\begin{aligned} \boldsymbol{p}'_i &= \tfrac{1}{8}(1\boldsymbol{p}_{i-1} + 6\boldsymbol{p}_i + 1\boldsymbol{p}_{i+1}) \\ \boldsymbol{p}'_{i+\frac{1}{2}} &= \tfrac{1}{8}(4\boldsymbol{p}_i + 4\boldsymbol{p}_{i+1}) \end{aligned} \quad (4)$$

$$\begin{aligned} \boldsymbol{p}'_0 &= \boldsymbol{p}_0 \\ \boldsymbol{p}'_{end} &= \boldsymbol{p}_{end} \end{aligned} \quad (5)$$

Results and comparisons of these two methods is presented in the result section at section 5. Even though the recursive formulation and the subdivision scheme are easy to grasp and implement in computer graphics it is not suitable for multiprocessing purposes. For simulation purposes the subdivision of guide hairs must be done per frame since the large amount of vertex points per hair might not fit onto CUDA block processors since there is an upper bound of threads per block.

2.3 Centroidal Voronoi Tessellations on Meshes

In order to place the guide hairs uniformly on the mesh a set of k randomly placed particles are needed. These particles are generated randomly by getting a face f and calculate its center f_c and area f_a of the face and compare it to a random variable f_r to see if it passes as a guide hair position or not. It will pass as a guide hair initial position if it does not satisfy equation 6.

$$\rho * f_c * f_a \leq f_r \quad (6)$$

where ρ is a density function. For uniformly distributed points the density function is set to $\rho = 1$.

2.4 Hair interpolation

After the guide hairs are subdivided into the discrete number of line segments an interpolation scheme is performed. To be able to generate new hairs random points have to be seeded out on the mesh to represent the hair root. A set of definitions are made to classify the line segments and points on the to-be interpolated strands. These definitions are made from guide hairs and their offsets o_i which are defined by:

$$\begin{aligned} \boldsymbol{o}_0 &= \boldsymbol{p}_0 \\ \boldsymbol{o}_i &= \boldsymbol{p}_i - p_{i-1} \end{aligned} \quad (7)$$

In equation 7 o_i denotes the offset along a hair strand and p_i are the vertex positions along that strand with i denoting the index from the root to the last point of the hair. The newly interpolated hair strand at the root position q is defined by:

$$\begin{aligned} \boldsymbol{p}'_0 &= \boldsymbol{q} \\ \boldsymbol{p}'_i &= \boldsymbol{p}'_{i-1} + o_i \end{aligned} \quad (8)$$

In equation 10, p'_i is the new position for a interpolated hair strand. Based on a influence distance d a wighted interpolation scheme can be applied where the distance d determines how many and which guide strands influence the final interpolated strand. Using the offsets from nearby guide strands are used together with a Gaussian eight kernel to obtain smooth interpolation. Any other kernel can be used to weight the interpolation. The Gaussian kernel used in the implementation is defined by:

$$w(o_0, p'_0) = \begin{cases} e^{-\frac{(o_0-p'_0)^2}{2\sigma^2}} & \|o_0 - p'_0\| < d \\ 0 & otherwise \end{cases} \quad (9)$$

In equation 9, $\sigma = 0.5d$ defines a cut-off based on the influence distance so that the Gaussian kernel does not have infinite width. A weighted mean is used to weight the neighbor guide strand offsets so that all the found neighbors influence the to-be interpolated hair strand according to equation 10, which results in:

$$\begin{aligned} \boldsymbol{p}'_0 &= \boldsymbol{q} \\ \boldsymbol{p}'_i &= \boldsymbol{p}'_{i-1} + \frac{\sum_{j=0}^{N} w(o_{j,0}, p_0) o_{j,i}}{\sum_{j=0}^{N} w(o_{j,0}, p_0)} \end{aligned} \tag{10}$$

N in equation 10 denotes the amount of found neighbor guides and weights are computed from the gaussian kernel in equation 9.

To find the nearby guide strands a search can be performed based on the euclidean distance being less then the influence distance d from the hair point to all of the guide hairs. This is the straightforward way of implementing a nearest neighbor search and has $O(n^2)$ complexity. As a preprocessing stage his can be acceptable but having in mind that a physical simulation of inter-hair collisions requires re-computation of nearby hair strands on a moving character this method will not result in interactive speeds. Speedup can be achieved by applying some spatial data structures such as Kd-trees, octrees etc. To account for modern hardware capabilities a GPU is used for this implementation. Nearest neighbor search is performed on CUDA. In CUDA a large number of threads can be spawned to run virtually in parallel. Threads are laid out on a grid of blocks which each have a number of threads, usually up to 512 threads and block size is of the magnitude of larger than 200 blocks. However in CUDA many accesses to global memory can actually result into a very inefficient algorithm therefore applying the naive approach will actually be slower on CUDA than on the CPU.

To give the GPU a fair representation in the implementation a different algorithm is used. A spatial data structure on CUDA can be created using a hash grid that is sorted. A hash grid does not occupy much memory since only integer values are saved on the memory instead of floating point xyz positions. The algorithm is described in a few steps. First a virtual grid is created from an axis aligned bounding box of the mesh and a grid cell size of the influence distance d defined above. The vertex positions are then divided by the cell size and rounded down to the next integer to obtain the points position in the grid. As a third step a hash function is applied to that position in the grid by a hash function[2] $hash(x, y, z) = (x * p1 \mathbf{xor} y * p2 \mathbf{xor} z * p3) \mathbf{mod} n$ where $p1, p2, p3$ are large prime numbers and n is the hash table size. The hash table is then sorted by the hash values together with the position indices using radix sort. In order for the sorted table to be of use another kernel on CUDA is used to compute the cell start index and end cell index for the hash table so that a simple loop can be made in the last step. As for the last step in the algorithm all of the hair positions are hashed and a nearest neighbor search can be performed in the neighboring 26 cells in the grid and compared to the values in the hash table. This search can be executed entirely parallel eg. every to-be interpolated hair can search for its neighbors in parallel. The resulting neighbors are then used in the final interpolation scheme given by equation 10.

3 Shading hair

3.1 The Kajiya-Kay hair shading model

This phenomenal method presented by Kajiya and Kay in 1989[3] simulates, in a simple but effective way, the two most important visual aspects of hair. The Kajiya-Kay hair shading model is divided into to components, a Lambertian diffuse component and a specular component. The diffuse component is simulated with a Lambertian cosine falloff function,

see equation 11, where the closer the tangent(T) of the hair is aligned with the light(L), the more light it receives.

$$\Psi_{Diffuse} = K_d sin(T, L) \tag{11}$$

The notation (T, L) is simply the angle between the tangent direction T, and the light direction L, and K_d is the diffuse coefficient.

For the anisotropic specular term seen in equation 12, Kajiya and Kay approximated the reflection of light hitting the hair to be in the perfect reflection angle. Also, due to that the normals on the small cylinder points in all directions around the cylinder the reflected light is independent of the azimuthal direction and thus forms a cone, see figure 1

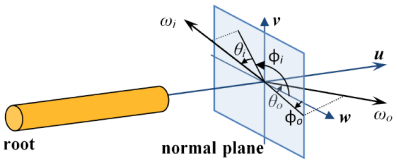


Figure 1: Scattering geometry notation and angles. Image courtesy of [4]

The specular intensity is given by equation 12,

$$\Psi_{specular} = Ks((T \cdot L)(T \cdot V) + sin(T, L)sin(T, L))^p \tag{12}$$

where K_s is a specular coefficient and p is the Phong exponent, which specifies the sharpness of the specular highlight. The final Kajiya kay shading model is seen in equation 13, where C_d and C_s are the diffuse and specular color respectively.

$$\Psi_{total} = C_d \Psi_{diffuse} + C_s \Psi_{specular} \tag{13}$$

Although this simple but phenomenal method captures the most significant key-effects of hair, the model is based on opaque cylinders and therefore it does not handle transmission and internal reflections which is very important key-aspects, especially when it comes to blond hair since it is more transparent than dark hair.

3.2 The Marschner hair shading model

In 2002 Marschner et al. [5] came up with a new hair shading method based on real hair observations. In comparison with the Kajiya-Kay hair shading method, Marscner's method also handles internal reflections(R) and transmissions(T). Marschner's hair shading method simulates three different scattering effects in hair fibers, namely reflection(R), transmission-transmission(T) and transmission-reflection-transmission (TRT), see figure 2. The model is based on the fact that the hair fibers are built up by small scales, much like the stem of a coconut tree, and it is decomposed into a sum of a set of azimuthal(N_t) and longitudinal(M_t) functions where $t \in (R, TT, TRT)$, see equation 14.

$$\Psi_{total} = \sum_i \frac{M_t(\theta_h) N_t(\theta_d, \phi)}{cos^2(\theta_d)} \tag{14}$$

In order to describe the angles in equation 14, a firm derivation of the hair fiber scattering geometry is in order. Figure 2 shows all notations of the hair fiber scattering geometry, where ω_i is the incoming light direction and ω_o is the outgoing light direction. These two directions are then expressed in spherical coordinates (θ, ϕ) during calculations. As seen in figure 2 the inclinations to the *normal plane* are denoted as θ_i and θ_o, and the azimuthal angles ϕ_i and ϕ_o are the angles in the normal plane. s Now, a description of the angles in equation 14 can properly be explained, where θ_h is the half angle $(\theta_i + \theta_o)/2$.

The azimuthal and longitudinal functions, N_t and M_t, which can both be precomputed and stored in lookup-tables. These functions will be described in these two following subsections.

3.2.1 The longitudinal scattering function M

The longitudinal function M is modeled as a normalized Gaussian distribution, here

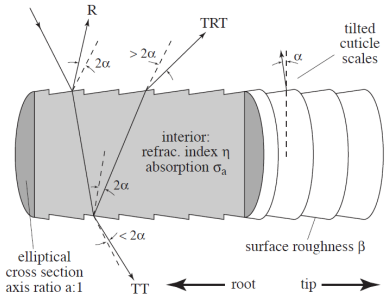


Figure 2: Mode of a hair fiber and the scattering paths illustrated. Image courtesy of [5]

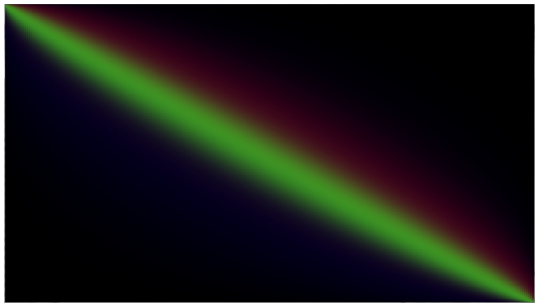


Figure 3: The lookup table of the longitudinal scattering functions M

denoted as the function $g(\alpha, \beta)$,where α is the mean and β is the standard deviation, which is applied to the different paths R, TT, TRT where the mean of M will be differently shifted for each lobe, see equation 15.

$$\begin{aligned} M_R(\theta_h) &= g(\theta_h - \alpha_R, \beta_R) \\ M_{TT}(\theta_h) &= g(\theta_h - \alpha_{TT}, \beta_{TT}) \\ M_{TRT}(\theta_h) &= g(\theta_h - \alpha_{TRT}, \beta_{TRT}) \end{aligned} \tag{15}$$

The mean, α, and standard deviation, β, values for the Gaussian distribution function for the paths R, TT and TRT are based on real measured values of hair. For description of how these values were derived, we refer the reader to Marschner's paper [5]. These functions M_R, M_{TT}, M_{TRT} are stored in a texture, here denoted M, along with a precalculated $cos(\theta_d)$ in the alpha channel, for later use to lookup values in the azimuthal textures N and N_TRT. The texture can be calculated during the initial face and later used as a lookup-table for computing the final hair shader. The M lookup-table is seen in figure 3.

3.2.2 The azimuthal scattering function N

The three modes of the azimuthal scattering function N are N_R, N_{TT}, N_{TRT} where N_R and N_{TT} are calculated using equation 16,

$$N_p(p, \phi) = \sum_r A(p, h(p, r, \phi)) \left| 2\frac{d\phi}{dh} h(p, r, \phi)) \right|^2 \tag{16}$$

where p is the different type paths(R, TT, TRT), between 0-2 and r is the number of roots. The function A is described in equation 16 where F is the fresnel equation, η' and η'' are two virtual indices of refraction, derived by applying snell's law and using Bravais's law, for a derivation on this and further reading, we refer the reader to Marschner's paper [5].

$$\begin{aligned} A(0, h) &= F(\eta', \eta'', \gamma_i) \\ A(p, h) &= (1 - F(\eta', \eta'', \gamma_i))^2 \\ & F(\tfrac{1}{\eta'}, \tfrac{1}{\eta''}, \gamma_t)^{p-1} T(\sigma'_a, h)^p \end{aligned} \tag{17}$$

The function T in equation 17 is The absorption contribution factor function $T(\sigma_a, h) = exp(-2\sigma_a(1 + cos(2\gamma_t)))$ where σ_a is the volume absorption coefficient per radius of the hair, $h = sin(\gamma_i)$ and γ_i is the angle of incidence.

The last component N_{TRT} is modeled in a different way in order to handle the transmissive-refractive-transmissive path but still uses the N_p function in equation 16 in the beginning. Marschner et al. [5] chose to display a pseudo-code of the N_{TRT} function as they thought it was the simplest way, and we will refer the reader to that pseudo-code instead of writing it again. The function has the following parameters, $N_{TRT}(\theta, \phi, \omega_c, k_G, \delta\eta', \delta h_M)$ where ω_c is

the width of the blur for the caustics, k_G adjusts the strength of the glints in the hair, $\delta\eta'$ is the range of η' and finally the δh_M is a limit on the caustic power often set to 0.5.

These three azimuthal scattering function components can be calculated during the initial stage and stored in two lookup textures where the first texture, denoted N stores the three color component N_{TT} function and the single value component N_R in the alpha channel. The third lookup-table N_{TRT} stores only its RGB-color components. To make it faster with almost the same quality by only having a single absorption value σ_a instead of three absorption values and store N_{TT} and N_{TRT} in the same texture.

4 Maya plug-in

The methods described in the above sections can be divided into their respective nodes in Maya. As it is stated in the Maya API every node can only compute values inside itself and never know about any other nodes in the hypergraph. The only thing that connects the nodes in Mayas DAG are input and output values of the nodes. An efficient Maya plug-in for hair generation includes a node that connects a mesh shape and computes the initial guide hair positions. This node is called *DistributeGuidePoints* which has attributes for the amount of guide hairs, iterations performed by the CVT and spline NURBS subdivision level for the guide hairs. This Node then has output connections to an interpolation node that generates new positions for the to-be interpolated hairs. This node is called *HairInterpolation* and has attributes to change the amount oh hairs and the influence distance. This node then outputs the new positions to a shape node that hold all the hairs that need to be rendered in the viewport. To illustrate the guide hairs a locator node is created for drawing of the *DistributeGuidePoints*. Then a hardware shading node is coupled to the shape node so that glsl shaders for both kajiya kay and marschner shaders. The resulting viewport image is seen in a figure below:

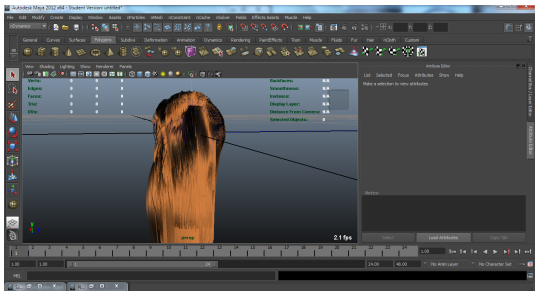


Figure 4: Screenshot from maya viewport when real-time rendering 30000 hairs using the marschner shader

5 Results

The following table shows how much time it takes to create 200 NURBS curves with 6 control points and 41 line segments:

Method	Result in ms
recursive	12.546
subdivision	5.653

Table 5 illustrates that the subdivision scheme is much faster in creating smooth NURBS curves. The drawback of both methods is that the implementation cannot be moved to CUDA for efficient computations firstly due to the recursive nature of the first algorithm and the backward dependency of the second algorithm. Backward dependency is not good since race conditions can occur writing and reading from the same array another thread is doing computations on.

The resulting approximate framerates in a stand-alone OpenGL program are shown below

Fps	amount of hairs	shader model
25	10000	marschner
36	10000	kajiya kay

The results in 5 illustrate that interactive framerates can be achieved by this implementation although many optimization possibilities have not been exploited.

The resulting shading models are shown in figures 5 and 6 where it can be seen that the marschner shader gives much more realistic results.

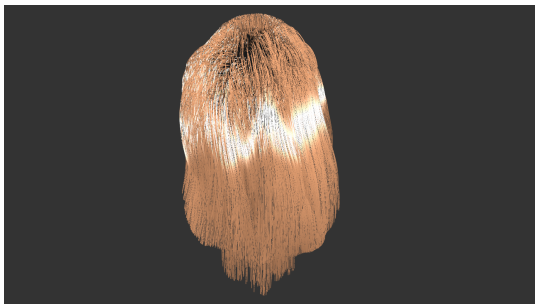


Figure 5: Image illustrating the result from Kajiya-kay shader

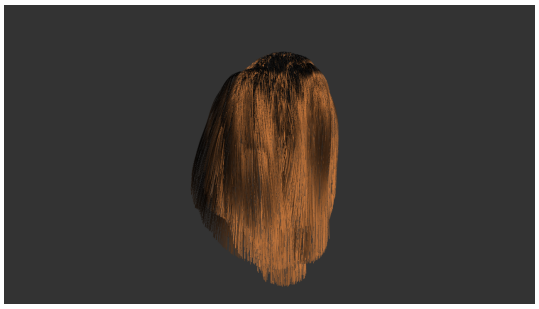


Figure 6: Image illustrating the result from marschner shader

The generated hair composited together with a backdrop is illustrated in figure 7

Figure 7: Image illustrating the result from marschner shader composited together with a backdrop

6 Discussion

The nearest neighbor search algorithm implemented using the virtual grid can generate bad results if the influence distance is not chosen well regarding to the amount of guide hairs and mesh size and curvature. The mesh curvature can be accounted for using different density functions in the CVT scheme and mean curvature values of the mesh which will move more points to a higher curvature thus more accurately accounting for uneven meshes.

Overall framerates for the hair shading can be improved using vertex buffer objects and multiple-texture single-framebuffer techniques instead of using multiple framebuffers for the lookup-tables in the marcher shader.

7 Future work

Future work would involve further discretization of the nodes in the plug-in to provide more control for the user adding painting possibilities on the hair strands so that manual grooming is possible. More interpolation nodes so that wet, frizz, curly and short hair styles are possible. Implementing a software rendering node so that the hair can be rendered in a final production quality image or video.

Depth map shading possibilities so that shadows and accurate depth is accounted for in the shaders. This can be done using the Deep opacity maps technique where interactive frame rates can be achieved. Dual scattering methods for extension of the marschner shader and to account for hair to hair light transport. Also implementing shading under environment lighting is a very good and useful property to have since it is used very often on scenes in computer graphics.

Hair object and inter hair collision detection for simulation possibilities of the hair. Also here interactive framerates can be achieved using a mass-spring model or

rigid body chains and a surface voxelization technique for object-hair collisions. Super helices method also handles hair to hair collisions but it is not possible to achieve real-time simulation. Using tessellation hardware could be another improvement on the simulation both for physical collision and hair interpolation stages.

References

[1] Q. Du, V. Faber, and M. Gunzburger, "Centroidal voronoi tessellations: Applications and algorithms," *SIAM Rev.*, vol. 41, pp. 637–676, December 1999.

[2] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," pp. 47–54, 2003.

[3] J. T. Kajiya and T. L. Kay, "Rendering fur with three dimensional textures," *SIGGRAPH Comput. Graph.*, vol. 23, pp. 271–280, July 1989.

[4] Z. Ren, K. Zhou, T. Li, W. Hua, and B. Guo, "Interactive hair rendering under environment lighting," *ACM Trans. Graph.*, vol. 29, pp. 55:1–55:8, July 2010.

[5] S. R. Marschner, H. W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan, "Light scattering from human hair fibers," *ACM Trans. Graph.*, vol. 22, pp. 780–791, July 2003.